jc531 U.S. PTO
09/585329

06/02/00

Assistant Commissioner for Patents
Washington, DC 20231

Atty. Dkt. 723-749

Date: June 2, 2000

This is a request for filing a new PATENT APPLICATION under Rule 53(b) entitled:
VARIABLE BIT FIELD COLOR ENCODING
without a filing fee and/or without an executed inventor's oath/declaration.
This application is made by the below identified inventor(s). Attached hereto are the following papers:

☒ An abstract together with
25 pages of specification and claims including
23 numbered claims and also attached is/are
10 sheets of accompanying drawings.
☐ This application is based on the following prior foreign application(s):

Application No.	**Country**	**Filing Date**

respectively, the entire content of which is hereby incorporated by reference in this application, and priority is hereby claimed therefrom.

☐ This application is based on the following prior provisional application(s):

Application No.	**Filing Date**

respectively, the entire content of which is hereby incorporated by reference in this application, and priority is hereby claimed therefrom.

☐ Certified copy/ies of foreign applications attached.
☐ This application is a ☐ continuation/☐ division/☐ continuation-in-part of application Serial No.   , filed   .
☐ Please amend the specification by inserting before the first line: --This application is a ☐ continuation/☐ division/ ☐ continuation-in-part of application Serial No.   , filed   , the entire content of which is hereby incorporated by reference in this application.--
☐ Please amend the specification by inserting before the first line: --This is a continuation of PCT application No.   , filed   , the entire content of which is hereby incorporated by reference in this application.--
☐ Please amend the specification by inserting before the first line: --This application claims the benefit of U.S. Provisional Application No.   , filed   , the entire content of which is hereby incorporated by reference in this application.--
☐ Preliminary amendment to claims (attached hereto), to be entered before calculation of the fee.
☐ Also attached.

Inventor:	Martin		Hollis	
	(first)	MI	(last)	(citizenship)
Residence: (city)	Cambridge		(state/country) United Kingdom	

Post Office Address: 51 St. Andrews Road, Cambridge, United Kingdom
(Zip Code) CB4 1DH

2.

Inventor:				
	(first)	MI	(last)	(citizenship)
Residence: (city)			(state/country)	

Post Office Address:
(Zip Code)

NOTE: FOR ADDITIONAL INVENTORS, check box ☐ and attach sheet with same information.

Address all future communications to NIXON & VANDERHYE P.C., 1100 North Glebe Road, 8th Floor, Arlington, Virginia 22201.

1100 North Glebe Road, 8th Floor
Arlington, Virginia 22201-4714
Telephone: (703) 816-4000
Facsimile: (703) 816-4100
RWF:ejs

NIXON & VANDERHYE P.C.
By Atty: Robert W. Faris, Reg. No. 31,352

Signature: _____

431967

U.S. PATENT APPLICATION

Inventor(s):   Martin Hollis

Invention: Variable Bit Field Encoding

SPECIFICATION

VARIABLE BIT FIELD ENCODING

FIELD OF THE INVENTION

This invention relates to computer graphics, and more particularly, to efficiently representing color image elements such as texels. Still more

5 particularly, the invention relates to a color image encoding format and associated encoding mode that provides higher resolution color information, or lower resolution color information and semi-transparency information.

BACKGROUND AND SUMMARY OF THE INVENTION

Many of us have seen films containing remarkably realistic dinosaurs, 10 aliens, animated toys and other fanciful creatures. Such animations are made possible by 3D computer graphics. A computer is used to model objects in three dimensions, and to display them on a screen such as your home television or computer screen. An artist can completely specify how each object will look as well as how it will change in appearance over time. The computer takes care of 15 performing the many millions of tasks required to make sure that each part of the moving image is colored just right based on how far away it is, the direction in which light strikes each of the many objects in the scene, the surface texture of each object, and many other factors.

Because of the complexity of the 3D graphics generation process, just a few 20 years ago computer-generated three-dimensional graphics was mostly limited to expensive specialized flight simulators, graphics workstations or supercomputers. The public saw the results of computer generated 3D graphics in movies and advertisements, but never actually interacted with the computers doing the 3D

graphics generation. All that has changed with the availability of relatively inexpensive 3D graphics platforms such as the Nintendo 64® and various 3D graphics cards available for personal computers. It is now possible to produce exciting 3D animations and simulations interactively in real time on relatively

5   inexpensive computer graphics systems in your home or office.

One goal of computer graphics is to provide the capability for a high degree of visual realism. This means that the computer ought to be able to model objects so they have visible characteristics just like real objects in the physical world. For example, to enable realistic lighting effects such as reflection, the computer should

10   keep track of which objects have shiny surfaces and which objects have dull surfaces. Another important characteristic the computer should be able to model is how opaque or transparent an object is. The computer should allow you to see through transparent objects such as windows, but not through opaque objects such as stone walls.

15   Many computer graphics system model the opacity (transparency) of surfaces using a technique called "alpha blending." Using this conventional technique, each image element is assigned an "alpha value" representing its degree of opacity. The colors of the image element are blended based on the alpha value - - allowing one object to appear to be visible through another object. A further

20   conventional technique called "alpha function" or "alpha test" can be used to discard an object fragment based on comparing the fragment's alpha value with a reference function or value. Alpha test may decide to not blend (i.e., to throw away) a potential part of an image because it is transparent and will therefore be invisible.

Alpha blending and alpha test are especially useful for modeling transparent objects such as water and glass. This same functionality can also be used with texture mapping to achieve a variety of effects. For example, the alpha test is frequently used to draw complicated geometry using texture maps on polygons --

5   with the alpha component acting as a matte. By way of illustration, a tree can be drawn as a picture (texture) of a tree on a polygon. The tree parts of the texture image can have an alpha value of 1 (opaque), and the non-tree parts can have an alpha value of 0 (transparent). In this way, the "non-tree" parts of the polygons are mapped to invisible (transparent) portions of the texture map, while the "tree"

10   portions of the polygon are mapped to visible (opaque) portions of the texture map.

The alpha component of a texture can be used in other ways -- for example, to cut holes or trim surfaces. As one example, an image of a cutout or a trim region can be stored in a texture map. When mapping the texture to the polygon

15   surface, alpha testing or blending can be used to cut the cutout or trimmed region out of the polygon's surface.

One interesting issue relates to the amount of alpha information that should be provided. In the real world, many objects are not completely transparent or completely opaque, but actually fall somewhere in between. For example, you

20   can't see through etched glass, but you can see some light shine through it. Etched glass is an example of an object that is neither entirely transparent or entirely opaque, but is instead semi-transparent or "translucent." Even objects we typically think of as being very transparent may not be entirely so but may instead be only semi-transparent. For example, pond water is relatively clear, but may have some

25   cloudiness to it. You can see a certain distance through pond water, but it

becomes increasingly opaque based on depth. Clouds, smoke and imaginary ghosts are other examples of semi-transparent objects you might want to model using a computer graphics system.

To model such semi-transparent objects, computer graphics systems in the past have used multi-bit alpha values that encode not just "opaque" and "transparent," but also varying degrees of semi-transparency. However, additional memory is needed to store an alpha component for each image element. The amount of additional memory required depends on the size of the image (i.e., the number of image elements) and the amount of alpha information to be stored for each image element. Storing multi-bit alpha components for each of thousands of image elements can substantially increase the amount of memory required. Even in systems with lots of memory, it may be desirable for performance reasons (i.e., reduced memory access time) to minimize the amount of memory required to store a given image.

To avoid locking application developers to a particular set of memory requirements and/or memory access times, one approach used in the past was to make the image element encoding mode of the computer graphics system programmable. Under this approach, the programmer could select between different color encoding modes as dictated by the characteristics of the particular image being generated at the time. For example, some systems allowed the programmer to choose between single-word and double-word color encoding formats. The programmer could choose a single-word RGB format for images requiring lower color resolution and no transparency capabilities, or a double-word RGBA format for images requiring higher color resolution and transparency. Speed performance might suffer somewhat if the double-word format were

selected (since two words instead of one need to be accessed for each image element), but this tradeoff might be worth it to enable more complex or interesting images to be generated.

While the approach of selecting between single-word RGB format and double-word RGBA format is very useful, it also has certain significant limitations. For example, in resource-constrained 3-D graphics systems such as 3-D home video games, it may be especially important as a practical matter to conserve memory usage and associated memory access time. This might mean, for example, that in the context of a real time interactive game, the programmer may rarely (if ever) have the luxury of activating the double-word RGBA mode because of memory space or speed performance considerations. In other words, even when using a system that provides an alpha mode, the game programmer may sometimes be unable to take advantage of it without degrading image complexity (e.g., number of textures) and/or speed performance.

One past proposed solution to this problem was to allocate a single bit of a single-word RGB color format for transparency. For example, if word length is 16 bits, five bits can be allocated to each of the three primary colors (red, green and blue) -- and the extra bit could be used for transparency (alpha). While this approach is certainly useful in terms of very efficient use of available memory, it has the limitation of providing only a binary (on/off) alpha value (i.e., either transparent or opaque). This prior approach therefore cannot provide visual effects requiring more alpha resolution (semi-transparency).

By way of further explanation, along edges of cutouts, trim regions, and certain texture mapped images, it may be desirable to provide an alpha component value that lies somewhere between transparent and opaque. This capability can

(coupled with conventional anti-aliasing techniques) smooth and soften transitions to increase realism. For example, in the real world, the edge(s) surrounding a cutout might not be an entirely sharp transition, but may instead have some smooth transition. Alpha blending based on a range of alpha

5 components modeling semi-transparency coupled with anti-aliasing (which smoothes out the "jaggies" in a digitally stepped surface) can be used to effectively model natural edge rounding. But this technique requires the ability to model semi-transparency, and does not work well if the alpha component is limited to a single "on/off" value.

10 Figures 1(a) and 1(b) help to illustrate this. Figure 1(a) shows an original texture of a tree on a black background. Figure 1(b) shows this same texture with one bit of alpha resolution. A one-bit alpha texture appears unnatural around the edges. This is a problem when the texture appears in front of any other objects, or indeed when the background to the texture is anything other than plain black. The

15 texture is pictured here on a plain white background.

We have realized that for many of the visual effects we wish to present in the context of video games and other 3D interactive applications, we want to be able to provide more than a single "on/off" (i.e., opaque or transparent) value, but we may not need a "full" resolution alpha component to accomplish our

20 objectives. For example, to provide smooth anti-aliased edges on cutouts, we may not need full 8-bit alpha resolution to provide visually pleasing effects. Some type of reduced resolution alpha encoding for semi-transparency (e.g., two or three bits of alpha to encode transparent, opaque, and two or six intermediate semi-transparency values) may be sufficient.

Figure 1(c) helps to illustrate this. Figure 1(c) shows the same tree texture as Figures 1(a) and 1(b), but using a multiple bit alpha value. The texture is pictured here on a plain white background. A multiple-bit alpha texture appears far smoother around the edges, whatever is behind it.

5   The present invention takes advantage of this observation by providing, in one particular implementation, a compact image element encoding format that selectively allocates bits on an element-by-element basis to encode multi-bit alpha resolution. This technique may be advantageously used to allocate encoding bits within some image elements for modeling semi-transparency while using those same bits for other purposes (e.g., higher color resolution) in other image elements not requiring a semi-transparency value (e.g., for opaque image elements). Applications include but are not limited to texture mapping in a 3D computer graphics system such as a home video game system or a personal computer.

In accordance with one aspect of the invention, a stored data element format representing a portion of an image includes a multi-bit alpha component field that may or may not be present in a particular instance of said format. The format includes a further portion encoding at least one color component. This portion has a first length if said multi-bit alpha component field is present, and has a second length greater than said first length if said multi-bit alpha component field is not

20   present.

In accordance with another aspect of the invention, a texture map includes a first texel encoded with a semi-transparency value and having first color resolution; and a second texel encoded without any semi-transparency value and having second color resolution greater than the first color resolution.

In accordance with a further aspect of the invention, a color image element encoding format comprises an indicator field indicating whether an instance of said format is capable of encoding semi-transparency. The format further includes at least one variable sized field encoding further information concerning the color image element. The at least one variable sized field has a first length if the indicator field indicates the format instance is incapable of encoding semi-transparency, and has a second length less than the first length if the indicator field indicates the format instance is capable of encoding semi-transparency.

In accordance with a further aspect of the invention, an image element encoding format includes a flag or other indicator that indicates whether the element has an associated a multi-bit alpha component. If the flag indicates that no alpha value is present, then the encoding format stores higher-resolution color information (e.g., five bits each of red, green and blue color information in one particular example). If, on the other hand, the indicator indicates that an alpha component is present, then the image element's color resolution is reduced (e.g., to four bits each of red, green and blue color information in one particular example), and the remaining bits are used to provide a multi-bit field to encode semi-transparency alpha information.

The present invention also provides a method of encoding an image element comprising specifying whether said image element will encode semi-transparency. If the specifying step specifies that said image element will encode semi-transparency, a set of plural bits within an encoding format is allocated to encode alpha. If the specifying step specifies that the image element will not encode semi-transparency, the set of plural bits is allocated to encode another characteristic of the image element (e.g., increased color resolution).

The present invention further provides an alpha component converter that converts between first and second resolutions of semi-transparency information, the converter quantizing or dequantizing first resolution semi-transparency information into a predetermined number of equal sized steps to form second

5   resolution semi-transparency information.

The ability to vary the bit encoding format on an image-element-by-image-element basis provides the potential for enhanced image quality by, for example, increasing the color resolution of those image elements not needing an alpha component. Opaque image elements can use the bits that may otherwise be used

10   for alpha encoding to achieve higher color resolution.

The variable bit field color encoding technique provided by the present invention is especially useful in encoding texture elements (texels) within a 3D graphics system. Such variable bit field color encoding can be used, for example, to provide a texture element multi-bit alpha component that allows smooth anti-

15   aliased edges on cutouts and in other instances where semi-transparency encoding is useful, without requiring the programmer to invoke a double-precision color encoding mode for all image elements with resulting doubling of the total amount of storage space required. Furthermore, this technique can be used to preserve higher color resolution across most of an image while degrading it only locally in

20   small image areas where semi-transparency is required. The loss of color resolution may not be noticeable in such small semi-transparent image areas.

BRIEF DESCRIPTION OF THE DRAWINGS

The file of this patent contains at least one drawing executed in color. Copies of this patent with color drawing(s) will be provided by the Patent and Trademark Office upon request and payment of the necessary fee.

5   These and other features and advantages may be better and more completely understood by referring to the following detailed description of presently preferred example embodiments in conjunction with the drawings, of which:

Figure 1(a) shows an example texture on a black background;

Figure 1(b) shows the Figure 1(a) texture with one bit alpha;

10   Figure 1(c) shows the Figure 1(a) texture with multi-bit alpha against a white background;

Figures 2A-2C show an example overall 3D interactive computer graphics system in which the present invention may be employed;

Figures 3A and 3B show an example variable bit encoding format;

15   Figure 4 shows an example texture memory map using the variable bit encoding format of Figures 3A and 3B;

Figure 5 shows an example mapping between reduced and higher resolution alpha encodings; and

Figure 6 shows an example texel component extraction multiplexing.

20   DETAILED DESCRIPTION OF PRESENTLY PREFERRED EXAMPLE EMBODIMENTS

Figure 2A is a schematic diagram of an overall example interactive 3D computer graphics system 100 in which the present invention may be practiced. System 100 can be used to play interactive 3D video games accompanied by

interesting stereo sound. Different games can be played by inserting appropriate storage media such as optical disks into an optical disk player 134. A game player can interact with system 100 in real time by manipulating input devices such as handheld controllers 132, which may include a variety of controls such as

5   joysticks, buttons, switches, keyboards or keypads, etc.

System 100 includes a main processor (CPU) 102, a main memory 104, and a graphics and audio coprocessor 106. In this example, main processor 102 receives inputs from handheld controllers 132 (and/or other input devices) via coprocessor 100. Main processor 102 interactively responds to such user inputs,

10   and executes a video game or other graphics program supplied, for example, by external storage 134. For example, main processor 102 can perform collision detection and animation processing in addition to a variety of real time interactive control functions.

Main processor 102 generates 3D graphics and audio commands and sends

15   them to graphics and audio coprocessor 106. The graphics and audio coprocessor 106 processes these commands to generate interesting visual images on a display 136 and stereo sounds on stereo loudspeakers 137R, 137L or other suitable sound-generating devices.

System 100 includes a TV encoder 140 that receives image signals from

20   coprocessor 100 and converts the image signals into composite video signals suitable for display on a standard display device 136 (e.g., a computer monitor or home color television set). System 100 also includes an audio codec (compressor/decompression) 138 that compresses and decompresses digitized audio signals (and may also convert between digital and analog audio signaling

25   formats). Audio codec 138 can receive audio inputs via a buffer 140 and provide

them to coprocessor 106 for processing (e.g., mixing with other audio signals the coprocessor generates and/or receives via a streaming audio output of optical disk device 134). Coprocessor 106 stores audio related information in a memory 144 that is dedicated to audio tasks. Coprocessor 106 provides the resulting audio

5   output signals to audio codec 138 for decompression and conversion to analog signals (e.g., via buffer amplifiers 142L, 142R) so they can be played by speakers 137L, 137R.

Coprocessor 106 has the ability to communicate with various peripherals that may be present within system 100. For example, a parallel digital bus 146

10   may be used to communicate with optical disk device 134. A serial peripheral bus 148 may communicate with a variety of peripherals including, for example, a ROM and/or real time clock 150, a modem 152, and flash memory 154. A further external serial bus 156 may be used to communicate with additional expansion memory 158 (e.g., a memory card).

15   **Graphics And Audio Coprocessor**

Figure 2B is a block diagram of components within coprocessor 106. Coprocessor 106 may be a single ASIC (application specific integrated circuit). In this example, coprocessor 106 includes a 3D graphics processor 107, a processor interface 108, a memory interface 110, an audio digital signal processor (DSP)

20   162, an audio memory interface (I/F) 164, an audio interface and mixer 166, a peripheral controller 168, and a display controller 128.

3D graphics processor 107 performs graphics processing tasks, and audio digital signal processor 162 performs audio processing tasks. Display controller 128 accesses image information from memory 104 and provides it to TV encoder

25   140 for display on display device 136. Audio interface and mixer 166 interfaces

with audio codec 138, and can also mix audio from different sources (e.g., a
streaming audio input from disk 134, the output of audio DSP 162, and external
audio input received via audio codec 138). Processor interface 108 provides a
data and control interface between main processor 102 and coprocessor 106.

5   Memory interface 110 provides a data and control interface between coprocessor
106 and memory 104. In this example, main processor 102 accesses main memory
104 via processor interface 108 and memory controller 110 that are part of
coprocessor 106. Peripheral controller 168 provides a data and control interface
between coprocessor 106 and the various peripherals mentioned above (e.g.,

10   optical disk device 134, controllers 132, ROM and/or real time clock 150, modem
152, flash memory 154, and memory card 158). Audio memory interface 164
provides an interface with audio memory 144.

Figure 2C shows a more detailed view of 3D graphics processor 107 and
associated components within coprocessor 106. 3D graphics processor 107

15   includes a command processor 114 and a 3D graphics pipeline 116. Main
processor 102 communicates streams of graphics data (i.e., display lists) to
command processor 114. Command processor 114 receives these display
commands and parses them (obtaining any additional data necessary to process
them from memory 104), and provides a stream of vertex commands to graphics

20   pipeline 116 for 3D processing and rendering. Graphics pipeline 116 generates a
3D image based on these commands. The resulting image information may be
transferred to main memory 104 for access by display controller 128 -- which
displays the frame buffer output of pipeline 116 on display 136.

In more detail, main processor 102 may store display lists in main memory

25   104, and pass pointers to command processor 114 via bus interface 108. The

command processor 114 fetches the command stream from CPU 102, fetches vertex attributes from the command stream and/or from vertex arrays in memory, converts attribute types to floating point format, and passes the resulting complete vertex polygon data to the graphics pipeline 116 for rendering/rasterization. A

5   memory arbitration circuitry 130 arbitrates memory access between graphics pipeline 116, command processor 114 and display unit 128.

As shown in Figure 2C, graphics pipeline 116 may include transform unit 118, a setup/rasterizer 120, a texture unit 122, a texture environment unit 124 and a pixel engine 126. In graphics pipeline 116, transform unit 118 performs a

10   variety of 3D transform operations, and may also perform lighting and texture effects. For example, transform unit 118 transforms incoming geometry per vertex from object space to screen space; transforms incoming texture coordinates and computes projective texture coordinates; performs polygon clipping; performs per vertex lighting computations; and performs bump mapping texture coordinate

15   generation. Set up/rasterizer 120 includes a set up unit which receives vertex data from the transform unit 118 and sends triangle set up information to rasterizers performing edge rasterization, texture coordinate rasterization and color rasterization. Texture unit 122 (which may include an on-chip texture memory 122') performs various tasks related to texturing, including multi-texture handling,

20   post-cache texture decompression, texture filtering, embossed bump mapping, shadows and lighting through the use of projective textures, and BLIT with alpha transparency and depth. Texture unit 122 outputs filtered texture values to the texture environment unit 124. Texture environment unit 124 blends the polygon color and texture color together, performing texture fog and other environment-

25   related functions.

Pixel engine 126 performs z buffering and blending, and stores data into an on-chip frame buffer memory. Graphics pipeline 116 may include one or more embedded DRAM memories to store frame buffer and/or texture information locally. The on-chip frame buffer is periodically written to main memory 104 for

5 access by display unit 128. The frame buffer output of graphics pipeline 116 (which is ultimately stored in main memory 104) is read each frame by display unit 128. Display unit 128 provides digital RGB pixel values for display on display 136.

Example Variable Bit Encoding Format

10 Figures 3A and 3B show an example image element variable bit encoding format. In the particular example shown, the format has a fixed length of 16 bits, but how those bits are allocated can vary on an instance-by-instance basis such that the same image map can use different encodings for different elements. In more detail, when the most significant bit (bit 15) is set, the remainder of the

15 format encodes higher resolution color information (for example, five bits each of red, green and blue color values) and defines an opaque image element. When the most significant bit is not set, the format provides lower resolution color information (for example, four bits each of red, green and blue) along with three bits of alpha information defining multiple levels of semi-transparency.

20 In more detail, the image element formats shown in Figures 3A and 3B may be among one of several different format options for the texture unit 122. Other texture formats may include, for example:

- sixteen-bit RGB (5 bits of red, 6 bits of green and 5 bits of blue),
- thirty-two bit RGBA (8 bits of each red, green, blue and alpha),
25 - four-bit intensity,

- eight-bit intensity,

- four-bit intensity and four-bit alpha,

- eight-bit intensity and eight-bit alpha, and

- various color indexed modes

5   The format shown in Figures 3A and 3B may be selected, for example, by specifying a *format* parameter in a graphics command directed to texture unit 122 for initializing a new texture object. Any given texture mapping will generally have a single overall format -- but in this particular example, the two alternate formats shown in Figures 3A and 3B are both encompassed by the same *format* parameter. The most significant bit (bit 15) within the format encoding specifies whether the particular instance of the format contains five bits each of red, green and blue information (RGB5); or alternatively, four bits each of red, green and blue plus three bits of alpha (RGB4A3).

Referring now specifically to Figure 3A, the most significant ("flag") bit 302 is set to 1, indicating an opaque, higher color resolution encoding. In the particular example shown, the encoding in this case comprises a total of sixteen bits, with:

- bit positions 14-11 occupied by a five-bit red color field 304;

- bit positions 9-5 occupied by a five-bit green color field 306; and

- bit positions 4-0 occupied by a five-bit blue color field 308.

In Figure 3B, in contrast, the flag 302 is set to 0 to indicate that the remaining 15 bits of that instance of the format encode lower resolution color information and a multi-bit alpha value. In this particular instance example:

- a three-bit alpha field 310 occupies bit positions 14-12;

- a four-bit red color field 312 occupies bit positions 11-8;

- a four-bit green color field 314 occupies bit positions 7-4; and

- a 4-bit blue color field 316 occupies bit positions 3-0.

Figure 4 shows an example portion of a texture map 320 stored in texture memory 122' using the variable bit format encoding scheme shown in Figures 3A and 3B. Figure 4 shows that different texels within the same texture map can have different ones of the encodings shown in Figures 3A and 3B. For example, certain texels along an edge of a cutout or the like requiring semi-transparency may be encoded using the Figure 3B format -- as indicated by the shaded texels shown in Figure 4. Other texels within the same texture map that are opaque as opposed to semi-transparent or transparent may use the Figure 3A encoding with higher color resolution and no transparency value -- as indicated by the unshaded elements of Figure 4. Of course, it would be possible to represent the entire texture map using either the Figure 3A format or the Figure 3B format if desired for uniformity or other reasons. The Figure 3B encoding in this example is capable of representing opaque texels (e.g., by setting the alpha field 310 to its maximum) or completely transparent texels (e.g., by setting alpha field 310 to its minimum value). However, if no texels are transparent or semi-transparent, then it might be preferable to use a different format without a flag field 302 (e.g., the RGB 565 field discussed above) to maximize memory usage efficiency.

Conversion Between Alpha Resolutions

One issue that arises when using the Figure 3B format is how to map or convert between higher resolution color and/or alpha component values and the lower resolution color and/or alpha component values shown in Figure 3B. As an example, the graphics pipeline shown in Figure 2C may provide eight-bit resolution for each of the red, green, blue and alpha channels. These 8-bit

component values may need to be compatible with the reduced resolution color and alpha information within the Figure 3A/3B encoding. Similarly, the frame buffer may be capable of storing image elements in higher resolution formats (e.g., 24-bit or 32-bit RGBA words), and it may be possible or desirable to generate

5   texture maps by copying image elements from the frame buffer to texture memory 122'. To accommodate such conversions, the example embodiment provides multiplexing and demultiplexing logic to map or convert between the lower resolution values shown in Figure 3B and the higher resolution values such as, for example, eight bits each of RGBA. Many different mappings are possible, but the

10   one we prefer for alpha is to provide eight evenly-spaced quantization levels as shown in Figure 5. A different number of quantization levels (e.g., nine) would also work, but for ease of use and compatibility with other formats, it may be most convenient to use eight levels as oppose to some other number. The following table shows an example mapping using eight equally spaced quantized levels:

A[7:0]	S	A[2:0]	Actual Value	Delta
0...31	32	0/7	0	
32...63	32	1/7	36	36
64...95	32	2/7	73	37+
96...127	32	3/7	109	36
128...159	32	4/7	146	37+
160...191	32	5/7	182	36
192...223	32	6/7	219	37+
224...255	32	7/7	255	36

Table I

15

As shown in the above table, "S" represents the size of the range that maps to one quantized representation. In this example, all range sizes are equal because the

quantized levels are equally spaced. The "delta" value D is the difference between dequantized values, with a "+" denoting a delta which is "high."

Figure 6 shows an example texel component extraction multiplexing technique provided to perform the mapping of Figure 5 and Table I. The Figure 6 multiplexing technique takes the Figure 3B format and maps it into eight bits each of alpha, red, green and blue. In this example, the mapping is performed in a straightforward fashion by simply repeating extracting bit patterns from the Figure 3B format, that is:

format	Texel alpha [7:0]	Texel red [7:0]	Texel green [7:0]	Texel blue [7:0]
0/A3/R4/G4/B4	[14:12] [14:12]] [14:13]	[11:8] [11:8]	[7:4] [7:4]	[3:0] [3:0]

While the invention has been described in connection with what is presently considered to be the most practical and preferred embodiment, it is to be understood that the invention is not to be limited to the disclosed embodiment. For example, the particular number of bits and/or the order of the bits described above could change depending upon the application. In addition, the variable bit encoding described above could be used as part of a color indexed value if desired. Also, the disclosed embodiment relates to a texture map encoding format, but the invention is not limited to texture representations. For example, pixels or other data items could benefit from the encoding provided by this invention. In addition, the applications provided by this invention are not limited by any means to generation of cutouts and trim surfaces. On the contrary, the invention is intended to cover various modifications and equivalent arrangements included within the scope of the appended claims.

WHAT IS CLAIMED IS:

1   1. A stored data element format representing a portion of an image, said

2   stored data element comprising:

3   a multi-bit alpha component field that may or may not be present in a

4   particular instance of said format; and

5   a portion encoding at least one color component, said portion having a first

6   length if said multi-bit alpha component field is present and having a second

7   length greater than said first length if said multi-bit alpha component field is not

8   present.

1   2. A stored data element format as in claim 1 further including a flag that

2   indicates whether said multi-bit alpha component field is present in a particular

3   instance of said format.

1   3. A texture map including:

2   a first texel encoded with a semi-transparency value and having first color

3   resolution; and

4   a second texel encoded without any semi-transparency value and having

5   second color resolution greater than said first color resolution.

1   4. A texture map as in claim 3 wherein said first and second texels each

2   further include a flag indicating whether said texel is encoded with a multi-bit

3   semi-transparency value.

1   5. A computer graphics system including a storage device storing plural

2   data elements corresponding to color image elements, said data elements each

3   including color information and an indicator field indicating whether or not said

4   data element provides a further, multi-bit field encoding semi-transparency.

1   6. A system as in claim 5 wherein said indicator field comprises a single bit

2   flag.

1   7. A system as in claim 5 wherein ones of said plural data elements that do

2   not encode semi-transparency use the space otherwise occupied by said multi-bit

3   field to encode said color information at higher resolution.

1   8. A system as in claim 5 wherein said color information encodes each of

2   the three primary colors.

1   9. A system as in claim 5 wherein each said data element has a 16-bit

2   length, and said indicator field comprises a single bit.

1   10. A system as in claim 5 wherein said multi-bit field consists of three bits.

1   11. A system as in claim 5 further including a data converter coupled to

2   said storage device, said data converter converting between said multi-bit semi-

3   transparency encoding and higher resolution alpha information.

1   12. A system as in claim 11 wherein said data converter quantizes or

2   dequantizes said higher resolution alpha information in equal steps.

1   13. A system as in claim 11 wherein said data converter quantizes or

2   dequantizes said higher resolution alpha information in eight equal steps.

1   14. A color image element encoding format comprising:

2   an indicator field indicating whether an instance of said format is capable of

3   encoding semi-transparency; and

4   at least one variable sized field encoding further information concerning

5   said color image element, said at least one variable sized field having a first length

6   if said indicator field indicates said format instance is incapable of encoding semi-

7   transparency, said at least one variable sized field having a second length less than

8   said first length if said indicator field indicates said format instance is capable of

9   encoding semi-transparency.

1   15. A color image element encoding format as in claim 14 wherein said

2   variable sized field encodes color information.

1   16. A color image element encoding format as in claim 14 wherein said

2   format includes a multi-bit alpha field if said indicator field indicates said format

3   instance is capable of encoding semi-transparency.

1   17. A color image element as in claim 14 wherein said color image element

2   encoding format encodes texels.

1   18. A method of encoding an image element comprising:

2   (a) specifying whether said image element will encode semi-transparency;

3   (b) if said specifying step specifies that said image element will encode

4   semi-transparency, allocating a set of plural bits within an encoding format to

5   encode alpha; and

6 (c) if said specifying step specifies that said image element will not encode

7 semi-transparency, allocating said set of plural bits to encode another

8 characteristic of said image element.

1 19. A method as in claim 18 wherein said image element comprises a texel.

1 20. A method as in claim 18 wherein said step (c) comprises encoding color

2 of said image element at higher resolution through use of said set of plural bits.

1 21. An alpha component converter that converts between first and second

2 resolutions of semi-transparency information, said converter quantizing or

3 dequantizing first resolution semi-transparency information into a predetermined

4 number of equal sized steps to form second resolution semi-transparency

5 information.

1 22. The alpha component converter of claim 21 wherein the number of

2 equal sized steps is eight.

1 23. A method of generating a cutout image within a 3D graphics system

2 having a 3D graphics pipeline that generates images based on polygons, said

3 graphics pipeline including a texture mapping unit storing a texture map including

4 a first set of semi-transparent colored texels, and a second set of opaque, colored

5 texels, said method comprising:

6 encoding each of said texels in a variable bit encoding format wherein a

7 predetermined bit field within said format is allocated to encode semi-transparency

8 of said first set of texels, and said predetermined bit field is allocated to encode

9 coloration of said second set of texels;

10   applying said texel encoding as a matte to a polygon using at least one alpha

11   operation, to generate a set of image elements; and

12   anti-aliasing said image elements.

ABSTRACT OF THE DISCLOSURE

A compact image element encoding format selectively allocates a bit field within the format to alternately encode either multi-bit alpha resolution or increased color resolution. This encoding technique may be advantageously used to allocate encoding bits to model semi-transparency while using those same bits for other purposes (e.g., higher color resolution) in instances where semi-transparency is not required (e.g., for opaque image elements). In one advantageous embodiment, the same encoding format can provide either RGB5 or RGB4A3, on an image-element-by-image-element basis. Applications include but are not limited to texture mapping in a 3D computer graphics system such as a home video game system or a personal computer.
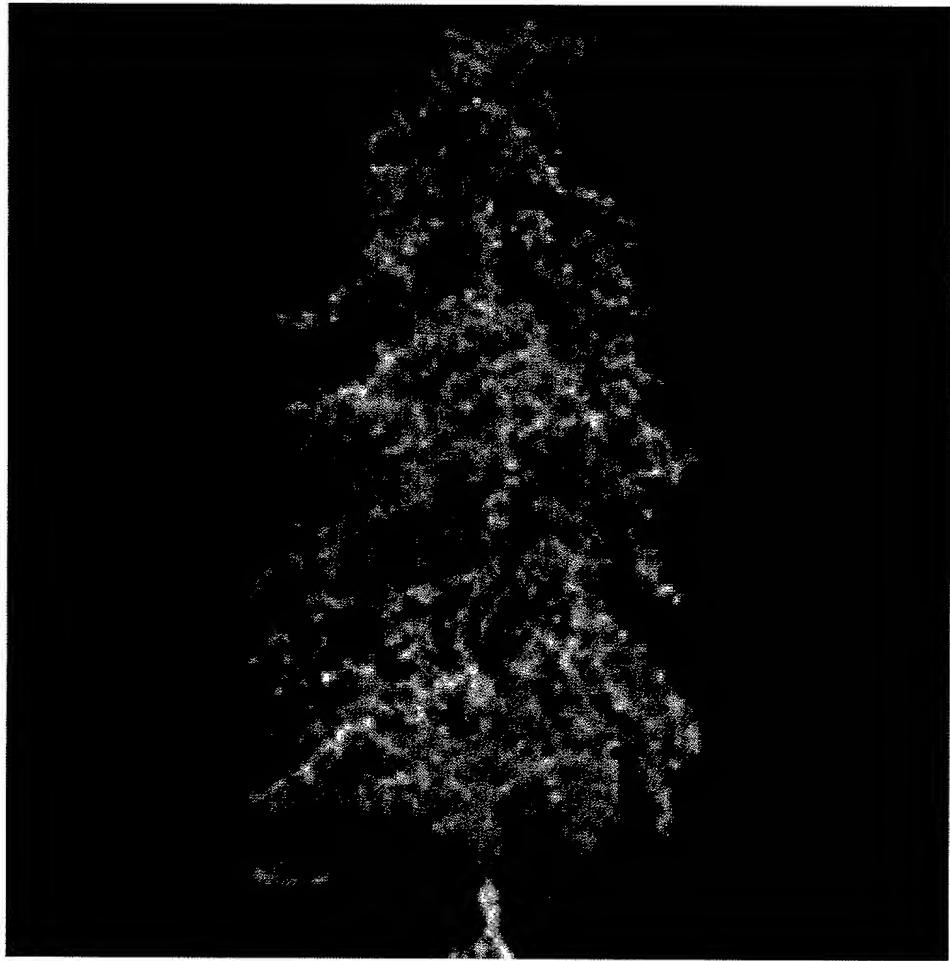
Fig. 1(a) Original texture
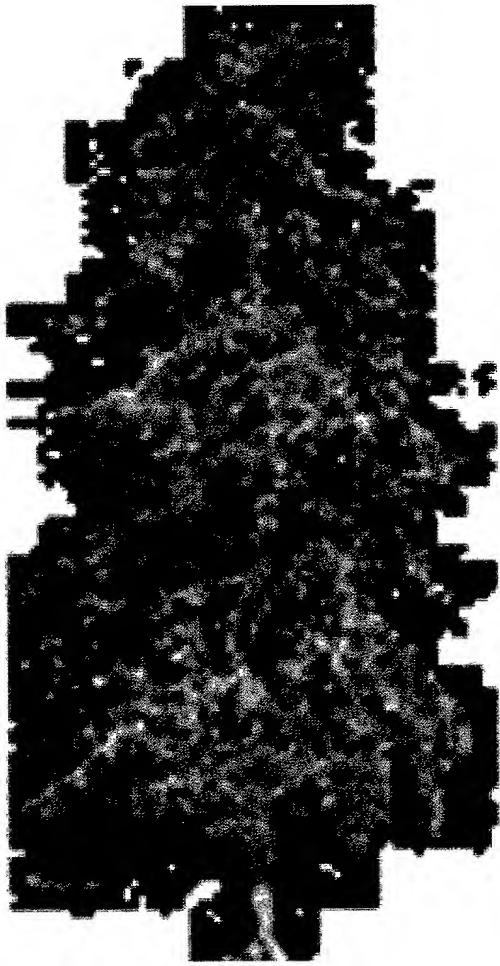
Fig. 1(b) Texture with one bit alpha
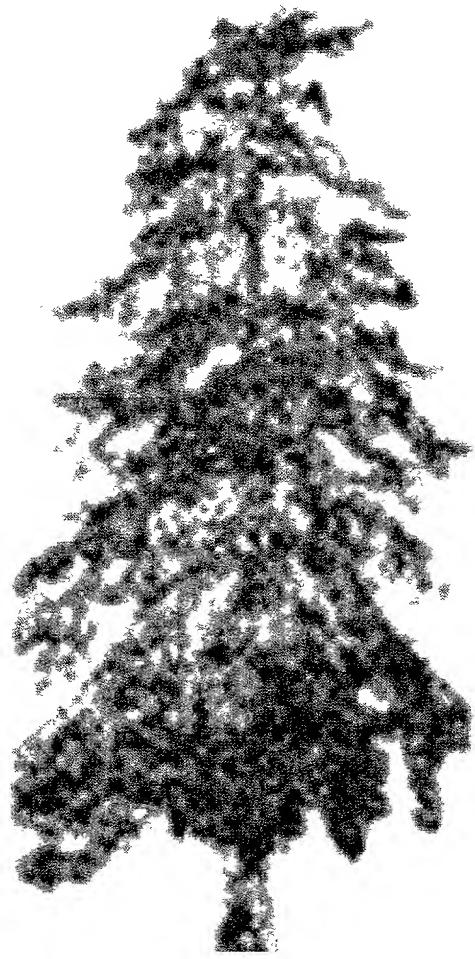
Fig. 1(c) Texture with multiple bit alpha

137L

137R

142L

142R

140

136

Audio in

Audio in/out

Memory
Card
158

TV
Encoder
140

Audio
codec
138

Hand
Controllers
132

CPU
102

Graphics and
Audio Coprocessor
106

Ext. SPI bus

SPI bus

PROM
RTC
150

Modem
152

Flash
Memory
154

156

148

100

DRAM
104

DRAM
104a

SDRAM
144

parallel bus

146

Optical Disk
Device
134

streaming
audio

Fig. 2A

Fig. 2B

106

TO/FROM
CPU
102

Processor
Interface
108

3D Graphics
Processor
107

Memory
Interface
110

TO/FROM
Memory
104

Video Out
to TV Encoder
140

Display
Controller
128

Audio
DSP
162

Audio
Memory I/F
164

TO/FROM
Audio Memory
144

Audio Inf
& Mixer
166

Audio
in/out

Peripheral
Controller
168

Disk Audio in
134

Disk
134

Controllers
132

ROM 150
Modem 152
Flash Memory 154

Memory
Card 158

Fig. 2C

US 2002/0067360 A1

200 →

Fig. 3(A)
RGB5

OR

Fig. 3(B)
RGB4A3

308

302

Byte 0

Byte 1

304 306

1	RED [4:0]	GREEN [4:0]	BLUE [4:0]
15	14 13 12 11 10 9	8 7 6 5	4 3 2 1 0

302

310

312

314

316

0	ALPHA [2:0]	RED [3:0]	GREEN [3:0]	BLUE [3:0]
15	14 13 12	11 10 9 8	7 6 5 4	3 2 1 0

Fig. 4

320

RGB4A3	RGB4A3	RGB4A3	RGB4A3
RGB4A3	RGB4A3	RGB4A3	RGB4A3
RGB4A3	RGB4A3	RGB4A3	RGB4A3
RGB4A3	RGB4A3	RGB4A3	RGB4A3
RGB5	RGB4A3	RGB4A3	RGB4A3
RGB5	RGB5	RGB4A3	RGB4A3
RGB5	RGB5	RGB5	RGB4A3
RGB5	RGB5	RGB5	RGB5
RGB5	RGB5	RGB5	

Fig. 5 Example mapping between 3-bit Alpha and 8-bit Alpha

15

Alpha   Red   Green   Blue

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

302 310 312 314 316

Texel Alpha [7:0]

| 14 | 13 | 12 | 14 | 13 | 12 | 14 | 13 |

Texel Red [7:0]

| 11 | 10 | 9 | 8 | 11 | 10 | 9 | 8 |

Texel Green [7:0]

| 7 | 6 | 5 | 4 | 7 | 6 | 5 | 4 |

Texel Blue [7:0]

| 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 |

Fig. 6   Example Texel Component Extraction Multiplexing